Laravel with Composer Managing Multi-Container Setup

Professional Laravel Development with Container Orchestration

Why Composer + Docker for Laravel?

Best of Both Worlds:

```
Composer Strengths:
✅ Laravel dependency management
✅ Automated scripts and tasks
✅ Package installation and updates
✅ Development workflow optimization

Docker Strengths:
✅ Environment consistency
✅ Service isolation
✅ Production deployment
✅ Infrastructure as code
```

The Synergy:

Composer manages Laravel complexity, Docker manages infrastructure complexity

Architecture Overview

Complete Laravel Stack:

```
[Browser] → [NGINX] → [PHP-FPM] → [Laravel App] → [MySQL]
                ↓          ↓              ↓            ↓
           [Static]   [Composer]    [Artisan]    [Redis Cache]
             Files    Dependencies   Commands     Session Store
```

Management Approach:

- **Composer**: Handles Laravel dependencies, scripts, and automation
- **Docker**: Provides consistent runtime environment
- **Laravel**: Application framework and business logic
- **Artisan**: Laravel's command-line interface

`setup.sh`

Instead of giving commands manually, we can use a shell script `setup.sh` (in `code/4_Docker/4. Containerizing Laravel with Docker`).

1. Copy the directory (code/4_Docker/4. Containerizing Laravel with Docker) to anything you choose to use, for example, `~/ex`.
2. Change to the copied directory (`cd ~/ex`).
3. Run the `setup.sh` to make a Laravel project (`bash setup.sh`).
    - It will generate a `hello` Laravel project in the directory.
4. You can change the project name with `bash setup.sh YOUR_PROJECT_NAME`.

For Windows Users

In case you use WSL2:

You can run this script without problem, but be sure to run `dos2unix setup.sh` when you have an EOL (End of Line) issue (such as $'\r': command not found).

In case you can't use WSL2:

1. Copy the directory.
2. Change to the copied directory.
3. Run each command line by line to get the same results.

What is this script?

- Creates a new web application (Laravel)
- Puts it in containers (Docker)
- Sets up a database
- Makes everything work together automatically

Like ordering a pizza 🍕: You just run one command, and everything is prepared for you!

📋 Step 1: Getting Ready

```bash
#!/bin/bash
set -e
```

What happens:

- The script starts
- `set -e` = "Stop if anything goes wrong" (safety first!)
- Sets up pretty colors for messages (Red, Green, Blue, Yellow)

Think of it as: Putting on your apron before cooking 👨‍🍳

📝 Step 2: Choose Your Project Name

```
if [ -z "$1" ]; then
    LARAVEL_DIR="hello"
else
    LARAVEL_DIR=$1
fi
```

What happens:

- If you don't give a name → uses "hello"
- If you do give a name → uses your name

Examples:

- `./setup.sh` → creates "hello" project
- `./setup.sh myapp` → creates "myapp" project

🏗️ Step 3: Create Laravel Project

```
composer create-project laravel/laravel "$LARAVEL_DIR"
```

What happens:

- Downloads Laravel (a web framework)
- Creates all the necessary files
- Like downloading and installing an app on your phone

Analogy: Building a house foundation 🏠

- Laravel = the basic structure of your web application

📁 Step 4: Copy Docker Files

```
cp -r "$SCRIPT_DIR/docker" .
cp "$SCRIPT_DIR/docker-compose.yml" .
```

What happens:

- Copies special Docker configuration files
- These tell Docker how to set up your containers

Analogy: Copying a recipe 📝

- The recipe tells Docker how to "cook" your application

🔧 Step 5: Update Configuration

```
sed -i "s|\./hello:|./${LARAVEL_DIR}:|g" docker-compose.yml
```

What happens:

- Updates the configuration to use YOUR project name
- Changes "hello" to whatever name you chose

Analogy: Writing your name on your homework 📝

- Makes sure Docker knows which project is yours

⚙️ Step 6: Environment Setup

```
cp "$SCRIPT_DIR/.env.docker" "$LARAVEL_DIR/.env"
```

What happens:

- Copies database connection settings
- Sets up environment variables (like settings)

Analogy: Programming your TV remote 📺

- Tells Laravel how to connect to the database

🐳 Step 7: Start Docker Containers

```
docker-compose up -d
```

What happens:

- Starts 3 containers:
  - **Web Server** (runs your Laravel app)
  - **Database** (stores your data)
  - **PHP** (runs your code)

Analogy: Starting a restaurant 🍽️

- Kitchen (PHP), Dining room (Web Server), Storage (Database)

⏳ Step 8: Wait and Check

```
sleep 15
# Check if MySQL is ready...
```

What happens:

- Waits for containers to start fully
- Tests database connection multiple times
- Like waiting for your computer to boot up

Why wait? Containers need time to initialize, just like apps on your phone

🔑 Step 9: Generate Security Key

```
docker exec laravel-php php artisan key:generate
```

What happens:

- Creates a unique security key for your app
- This key encrypts sensitive data

Analogy: Creating a password for your app 🔐

- Every Laravel app needs its own unique key

🗄️ Step 10: Set Up Database

```
docker exec laravel-php php artisan migrate
```

What happens:

- Creates database tables
- Sets up the database structure

Analogy: Building shelves in a warehouse 📦

- Creates organized storage for your data

🎉 Step 11: Success Message

```
echo "🌐 Your Laravel application is ready!"
echo "   Visit: http://localhost:8080"
```

What you get:

- A working web application
- Accessible at `<http://localhost:8080>`
- Useful commands to manage your app

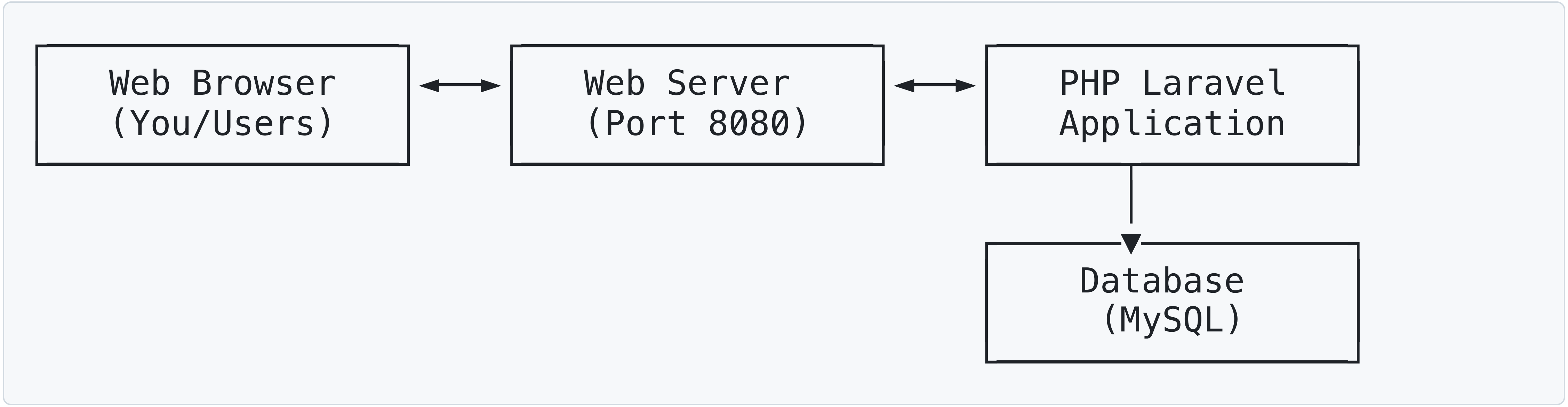
🧠 What Actually Happened?

Before: Nothing ❌

After: Complete web development environment ✅

1. **Laravel App** → Your web application code
2. **Database** → Stores your data (users, posts, etc.)
3. **Web Server** → Serves your app to browsers
4. **All Connected** → Everything talks to each other

📊 The Big Picture



🛠️ Useful Commands You Get

After setup, you can use:

- `composer run stop` → Stop everything
- `composer run shell` → Access the app container
- `composer run artisan` → Run Laravel commands
- `docker-compose logs -f` → See what's happening

Like TV remote buttons 📺 - each does something specific!

Configuration Files

To automate the deployment of a Laravel project using Docker, we need multiple configuration files.

```
📁 Laravel Docker Project
├── 🔧 .env.docker          ← Laravel app settings
├── 🐳 docker-compose.yml   ← Container coordinator
├── 📦 composer.json        ← PHP package manager
├── 🔧 .env                 ← composer settings
└── 📁 docker/              ← Container recipes
    ├── 🐘 php/
    │   ├── Dockerfile      ← PHP container recipe
    │   └── php.ini         ← PHP development settings
    ├── 🌐 nginx/
    │   ├── Dockerfile      ← Web server recipe
    │   └── nginx.conf      ← Web server settings
    └── 🗄 mysql/
        ├── Dockerfile      ← Database container recipe
        └── init.sql        ← Database initialization
```

🔧 File #1: `.env.docker` (Laravel Application Settings)

This will be copied into the Laravel project directory as `.env`.

What is it? Environment variables file for Laravel

Think of it as: Your app's preferences file

Key sections:

```
APP_NAME="Laravel Docker App"     # What to call your app
APP_DEBUG=true                    # Show errors (helpful for learning)
APP_URL=http://localhost:8080     # Where people find your app
```

Like: Setting your name, phone wallpaper, and notification preferences

🗄️ Database Settings in `.env.docker`

```
DB_CONNECTION=mysql           # Type of database (MySQL)
DB_HOST=mysql                # Where the database lives (container name)
DB_PORT=3306                 # Database door number
DB_DATABASE=laravel          # Database name
DB_USERNAME=laravel          # Username to access database
DB_PASSWORD=laravel_password # Password to access database
```

Laravel knows the database information from these settings.

Analogy: Like giving your app the address, apartment number, and keys to the database building

🔧 Understanding Two Types of Environment Variables

📱 Laravel App Configuration (`.env.docker` → `hello/.env`)

Purpose: Tell Laravel **HOW** to connect to MySQL

Used by: Laravel application code at runtime

```
# These variables tell Laravel:
DB_HOST=mysql                # "Connect to container named 'mysql'"
DB_DATABASE=laravel          # "Use database called 'laravel'"
DB_USERNAME=laravel          # "Login as user 'laravel'"
DB_PASSWORD=laravel_password # "Use this password"
```

Without using Docker, we use the IP address to specify the DB_HOST.

```
DB_HOST=127.0.0.1 # <--
DB_PORT=3306
```

📨 Other Services in `.env.docker`

```
MAIL_MAILER=log                # How to send emails (log = fake for testing)
CACHE_DRIVER=database          # Where to store temporary data
SESSION_DRIVER=file            # How to remember logged-in users
```

Think of it as: Telling your app which postal service to use, where to put sticky notes, and how to remember friends

🐳 File #2: `docker-compose.yml`

The Orchestra Conductor

What is it? Tells Docker how to run multiple containers together

Think of it as: A conductor's sheet music 🎼

Three main "musicians":

- 🌐 **nginx** (Web Server) - Greets visitors
- 🐘 **php** (Application) - Runs Laravel code
- 🗄 **mysql** (Database) - Stores data

🌐 Nginx Service Configuration (1/3)

```yaml
nginx:
  build: ./docker/nginx        # Use our custom recipe
  container_name: laravel-nginx # Name this container
  ports:
    - "8080:80"                # Map port 8080 (outside) to 80 (inside)
  volumes:
    - ./hello:/var/www/html    # Share Laravel code with container
  depends_on:
    - php                      # Wait for PHP to start first
```

Analogy: Setting up a receptionist desk that forwards visitors to the correct department

🐘 PHP Service Configuration (2/3)

```
php:
  build: ./docker/php          # Use our PHP recipe
  container_name: laravel-php  # Name this container
  volumes:
    - ./hello:/var/www/html    # Share Laravel code
  depends_on:
    mysql:
      condition: service_healthy # Wait for database to be ready
```

In the volumes, the Laravel project (hello) is mapped to internal /var/www/html, and other files are copied into the container.

```
COPY php.ini /usr/local/etc/php/conf.d/laravel.ini
```

Like: Setting up a chef who needs access to the kitchen (code) and waits for ingredients (database) to arrive

🗄️ MySQL Service Configuration (3/3)

```yaml
mysql:
  build: ./docker/mysql                 # Use our custom MySQL recipe
  container_name: laravel-mysql
  environment:                           # ✅ Pass environment variables from host
    - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
    - MYSQL_DATABASE=${MYSQL_DATABASE}
    - MYSQL_USER=${MYSQL_USER}
    - MYSQL_PASSWORD=${MYSQL_PASSWORD}
  volumes:
    - mysql-data:/var/lib/mysql
    - ./docker/mysql/init.sql:/docker-entrypoint-initdb.d/init.sql  # Database initialization
  networks:
    - laravel-network
```

From the environment variablse, Docker automatically runs the equivalent SQL.

```sql
CREATE DATABASE IF NOT EXISTS student_api;
CREATE USER 'student'@'%' IDENTIFIED BY 'secret';
GRANT ALL PRIVILEGES ON student_api.* TO 'student'@'%';
```

healthcheck

```yaml
healthcheck:
  test: ["CMD-SHELL", "mysqladmin ping -h 127.0.0.1 -u root -p$MYSQL_ROOT_PASSWORD --silent"]
  interval: 10s
  timeout: 5s
  retries: 5
  start_period: 30s
```

Like: Setting up a secure filing cabinet with specific locks and keys 🗃️🔐

🗄️ Understanding MySQL Health Checks

What is a Health Check?

A health check is like asking "Are you ready to work?" before giving someone a task.

Breaking it down:

- `mysqladmin ping`: Tests if MySQL can accept connections
- `-h 127.0.0.1`: Connect to localhost (more reliable than "mysql")
- `-u root`: Use root user credentials
- `-p$MYSQL_ROOT_PASSWORD`: Use password from environment variable
- `--silent`: Don't show verbose output

Note:

- `CMD-SHELL` allows shell variable expansion (`$MYSQL_ROOT_PASSWORD`)
- `start_period: 30s` gives MySQL time to initialize without marking it unhealthy
- This prevents Laravel from trying to connect before MySQL is truly ready

Like: Making sure the librarian is at their desk before asking for a book!

🐳 Container Configuration by Docker Compose (`.env` file)

Purpose: Tell MySQL container **WHAT** to create

Used by: Docker Compose when starting containers

```
# These variables tell the MySQL container:
MYSQL_DATABASE=laravel          # "Create database called 'laravel'"
MYSQL_USER=laravel             # "Create user called 'laravel'"
MYSQL_PASSWORD=laravel_password # "Give user this password"
MYSQL_ROOT_PASSWORD=root        # "Set root password"
```

The PHP (Laravel) container will use the MySQL database generated from the MySQL container.

⚠️ Warning!

- Composer doesn't create a MySQL database.
- MySQL database is created INSIDE the MySQL container, not on your local machine.
- .env configures the container to create the database inside the container.
- No MySQL installed on your local machine - everything runs in containers.

🔧 Hint for backing up the DB!

The MySQL database is created INSIDE the MySQL container, so we need to keep a backup.

```
# Create SQL backup file on your local machine
docker exec laravel-mysql mysqldump -u root -proot laravel > backup.sql

# Restore from backup
docker exec -i laravel-mysql mysql -u root -proot laravel < backup.sql
```

We can add a backup script to `composer.json` (will be explained in the next section).

```
"scripts": {
  "backup": "docker exec laravel-mysql mysqldump -u root -proot laravel > backups/$(date +%Y%m%d_%H%M%S).sql",
  "restore": "docker exec -i laravel-mysql mysql -u root -proot laravel"
}
```

📦 File #3: `composer.json`

PHP Package Manager & Shortcuts

What is it? Tells PHP what libraries to install and provides shortcuts

Think of it as: Your app's shopping list and remote control 📺 🛒

Two main parts:

1. **Dependencies** - What libraries to download
2. **Scripts** - Shortcut commands

📚 Dependencies in `composer.json`

```
"require": {
  "php": "^8.1",                  # Need PHP version 8.1 or newer
  "laravel/framework": "^10.10" # Need Laravel version 10.10+
}
```

Like: Telling your assistant "I need a car (PHP 8.1+) and GPS system (Laravel 10.10+)"

Result: Composer automatically downloads and installs everything you need!

🎮 Scripts (Shortcuts) in `composer.json`

```
"scripts": {
  "start": "docker-compose up -d",      # Start everything
  "stop": "docker-compose down",        # Stop everything
  "shell": "docker exec -it laravel-php bash", # Access PHP container
  "artisan": "docker exec laravel-php php artisan" # Run Laravel commands
}
```

Usage: `composer run start` instead of typing long Docker commands (such as `docker-compose up -d`)!

Like: TV remote buttons instead of manually adjusting settings

🐘 File #4: `docker/php/Dockerfile`

Recipe for PHP Container

What is it? Instructions to build a custom PHP container

Think of it as: Recipe for a specialized chef 👨‍🍳

Steps:

1. Start with basic PHP (base chef)
2. Install extra tools (give the chef special skills)
3. Configure settings (teach Chef your preferences)

🔧 PHP Dockerfile Breakdown

```
FROM php:8.2-fpm                       # Start with PHP 8.2
RUN apt-get install git curl zip... # Install system tools
RUN docker-php-ext-install pdo_mysql # Add database connection ability
COPY --from=composer /usr/bin/composer # Add package manager
WORKDIR /var/www/html                  # Set kitchen location
RUN chown -R www-data:www-data...      # Set proper permissions
```

Result: A PHP container that can run Laravel and connect to MySQL! 🚀

🐘 What is `php:8.2-fpm`?

- It's a **Docker base image** built on **Debian/Ubuntu (Linux)**.
- Already includes:
    - PHP 8.2 (compiled & ready)
    - FPM (FastCGI Process Manager)
- Maintained officially on **Docker Hub**.

✅ Benefits of Using `php:8.2-fpm`

- **Preconfigured PHP**: Maintained by Docker & PHP team
- **Security updates**: Patched automatically in official image
- **Consistency**: Same environment across all developers
- **Less code**: No need to build PHP yourself

🌐 File #5: `docker/nginx/Dockerfile`

Recipe for Web Server Container

What is it? Instructions to build a custom web server

Think of it as: Recipe for a specialized receptionist 💁‍♀️

```
FROM nginx:alpine               # Start with a lightweight web server
COPY nginx.conf /etc/nginx/... # Give it our custom rules
RUN mkdir -p /var/www/html...  # Create workspace
RUN chown -R nginx:nginx...    # Set permissions
EXPOSE 80                      # Open door on port 80
```

What is `nginx:alpine`?

It means the Nginx container uses Alpine Linux.

- Smaller image size
- Faster builds & deployments
- More secure containers
- Efficient and portable base for web apps

🗄️ File #6: `docker/mysql/Dockerfile`

Recipe for Database Container

What is it? Instructions to build a custom MySQL container

Think of it as: Recipe for a specialized data manager 🗃️

```
FROM mysql:8.0

# Allow remote connections (for Docker networking)
ENV MYSQL_ROOT_HOST=%

# Minimal configuration for MySQL 8.0 compatibility
RUN echo '[mysqld]' > /etc/mysql/conf.d/laravel.cnf \
    && echo 'default-authentication-plugin=mysql_native_password' >> /etc/mysql/conf.d/laravel.cnf

EXPOSE 3306
```

Benefits of using Dockerfile

- **No Hardcoded Values**: Environment variables come from docker-compose.yml
- **Laravel Compatibility**: Uses `mysql_native_password` authentication
- **Flexible Configuration**: Same Dockerfile works for dev/staging/production
- **Security**: Credentials managed externally, not baked into image

Result: A clean, configurable MySQL container that gets settings from external sources!

🐬 What is `mysql:8.0`?

- An **official MySQL Docker image** based on Linux (Debian/Ubuntu).
- Includes:
  - MySQL 8.0 server pre-installed
  - Default configuration files
  - Scripts to initialize users/databases

🗄️ File #7: `docker/mysql/init.sql`

Database Initialization Script

What is it? SQL script that runs automatically when the container first starts

Think of it as: Setting up the filing system in your new office 🗂️

```sql
-- Simple Laravel database initialization
-- ensure the database exists - Laravel migrations will handle the rest
USE laravel;

-- Simple test to ensure the database is ready
SELECT 'Database ready for Laravel!' as status;
```

🎇 How Initialization Works:

1. **First Container Start**: MySQL runs all `.sql` files in `/docker-entrypoint-initdb.d/`
2. **Volume Mounting**: `./docker/mysql/init.sql:/docker-entrypoint-initdb.d/init.sql`
3. **Automatic Execution**: No manual intervention needed
4. **One-Time Only**: Only runs on a fresh database (not on restarts)

Perfect for: Database seeding, initial user setup, or schema creation!

⚙️ File #8: `docker/nginx/nginx.conf`

Web Server Rules & Behavior

What is it? Detailed instructions for how the web server should behave

Think of it as: A receptionist's detailed job description 📋

Key responsibilities:

- Listen on port 80 for visitors
- Send PHP files to the PHP container for processing
- Serve static files (images, CSS) directly
- Add security headers to protect users

🌐 Nginx Configuration Sections

```
server {
  listen 80;                          # Listen on port 80
  root /var/www/html/public;          # Laravel's front door

  location / {                        # For regular pages
    try_files $uri /index.php;        # Try file, then Laravel
  }

  location ~ \.php$ {                 # For PHP files
    fastcgi_pass laravel-php:9000;    # Send to PHP container
  }

  location ~* \.(css|js|png)$ {       # For static files
    expires 1y;                       # Cache for 1 year
  }
}
```

🐘 File #9: `docker/php/php.ini`

PHP Engine Settings

What is it? Detailed settings for how PHP should run

Think of it as: Engine tuning for a race car 🏎️

Key categories:

- **Memory & Speed** - How much RAM to use, how long scripts can run
- **File Uploads** - How big files can be uploaded
- **Security** - What PHP is allowed to do
- **Error Reporting** - How to show problems (helpful for learning!)

🔧 Important PHP Settings Explained

```
memory_limit = 512M                # Use up to 512MB RAM
max_execution_time = 300           # Scripts can run for 5 minutes max
upload_max_filesize = 50M          # Allow 50MB file uploads
display_errors = On                # Show errors (good for learning!)
```

Analogy: Like setting limits on your car - max speed, fuel tank size, safety features 🚗

For Students: `display_errors = On` means you'll see helpful error messages!

Building for Production

The `docker/php/` also has `php-production.ini` and `Docker.prod` files for production release.

Development vs Production Dockerfiles

Aspect	Development	Production
Build Strategy	Single stage	Multi-stage build
Code Copying	Volume mounted	Copied into image
Dependencies	All (dev + prod)	Production only
Optimizations	None	Laravel caching, composer optimize
Security	Root user	Non-root user (www-data)
Size	Larger	Smaller, optimized

Development vs Production PHP Configuration

Setting	Development	Production	Why Different?
Memory Limit	512M	256M	Production needs resource control
Execution Time	300s	60s	Prevent long-running scripts in prod
Error Display	ON	OFF	Don't expose errors to users
File Upload Size	50M	10M	Security and resource control
OPcache Validation	Enabled	Disabled	Performance vs development flexibility
Dangerous Functions	Allowed	Disabled	Security hardening

🔗 How All Files Work Together

```
1. docker-compose.yml reads Dockerfiles
   ↓
2. Dockerfiles build containers using .ini/.conf files
   ↓
3. Containers start up with custom settings
   ↓
4. Laravel app uses the .env file (copied from the .env.docker) for configuration
   ↓
5. composer.json provides easy commands to manage everything
```

Like: Building a complete restaurant with specialized staff, each knowing their job! 🍽️

🎯 Real-World Analogy: Restaurant Setup

File	Restaurant Role	What It Does
`docker-compose.yml`	🎬 **Manager**	Coordinates everyone
`nginx/Dockerfile`	💁 **Hostess Recipe**	How to train hostess
`nginx.conf`	📋 **Hostess Rules**	Where to seat guests
`php/Dockerfile`	👨‍🍳 **Chef Recipe**	How to train chef
`php.ini`	⚙️ **Kitchen Settings**	Oven temp, tools available
`mysql/Dockerfile`	🗃️ **Manager Recipe**	How to train data manager
`mysql/init.sql`	🗂️ **Filing System**	How to organize data storage
`.env.docker`	📞 **Contact Info**	Phone numbers, addresses
`composer.json`	📱 **Speed Dial**	Quick commands

💡 Why So Many Files?

Separation of Concerns 🎯

- Each file has ONE specific job
- Easy to modify without breaking others
- Different team members can work on different parts

Like: Having separate instruction manuals for:

- 📺 TV remote (not mixing with microwave instructions)
- 🚗 Car manual (separate from house manual)
- 👕 Washing machine (different from dishwasher)

🔍 What Happens When You Run `setup.sh`?

1. **Creates Laravel project** 📁
2. **Copies all config files** 📋
3. **Updates docker-compose.yml** with your project name 🔧
4. **Docker reads Dockerfiles** → Builds containers 🏗️
5. **Containers use .ini/.conf files** → Custom settings ⚙️
6. **Laravel uses .env.docker** → Connects to database 🔗
7. **composer.json provides shortcuts** → Easy management 🎮

🛠️ Practical Tips for Students

To modify settings:

- Change app name → Edit `.env.docker`
- Add PHP extensions → Edit `php/Dockerfile`
- Change web server behavior → Edit `nginx.conf`
- Add shortcuts → Edit `composer.json` scripts

Golden Rule: 🥇 Always test changes with `composer run start`

Safety Tip: 🚨 Make backups before changing configuration files!

📚 Key Takeaways

✅ **Configuration files are instruction manuals**
✅ **Each file has a specific purpose**
✅ **They work together like a team**
✅ **Changes in one file can affect others**
✅ **Understanding these makes you a better developer**

Bottom Line: These files turn a complex multi-service application into something manageable and maintainable! 🎯